CIFAR-10 excercise

Michał Wiktor

Summary

The report describes two attempts classify 10 categories of figures given in CIFAR-10 dataset. Two approaches: shallow classifier based on simple neural network and transfer learning is presented. The former was unsuccessful, due to improper choice of fixed convolution filters. The latter resulted in 88% accuracy, which is not the state of the art result, but undoubtedly positively verifies the transfer learning approach and this specific implementation.

1 Problem overview

The task was to write a number of scripts for image recognition, specifically CIFAR-10 dataset. It contains 60000 color images of size 32 by 32 grouped into 10 categories. The reference set is divided into 5 training and one test set, each of size 10000 images. The suggested methods for performing the task was transfer learning and building the shallow classifier.

Before these two methods will be described, a few words should be written about image recognition, at general. The state of the art method is so-called convolutional neural network. The canonical neural network formulation assumes that the data are transmitted from input layer, through a number of hidden layers to the output layer, where each of the layers can be equivalent i equations of the form

$$out_i = f\left(\sum_{j=1}^{N} a_{i,j} in_j + b_i\right) \tag{1}$$

where in are the input signals out are outputs, b is a bias defined for each output neuron separately and f is (usually nonlinear) activation function. Parameters $a_{i,j}$ are supposed to be found during training (optimization) process.

In case of convolutional neural network (CNN), before the data are passed through layers equivalent to (1), they are and passed through nonlinear and convolution filters. The coefficients of the filters are also subject to be optimized. What is worth to note, the number of possible states within CNN layer is larger than number of states in analyzed picture, which make system of neurons strongly overdetermined.

2 Preliminary tasks

Before any image processing can be done, the data must be downloaded and appropriately prepared for further analysis. Image datasets are relatively big, thus the data should be downloaded once and kept for further processing. The **cifar_init.py** script, responsible for downloading CIFAR-10 dataset, checks if the data are present in current directory, and downloads is if it is necessary. The script calls external commands: **wget** and **tar** so it must be run in UNIX-like environment (Linux,BSD,Darwin,OS X).

Figure 1: Example figures grouped into 10 categories. Plot produced by **cifar_init.py**.

3 Shallow Classifier

The shallow classifier, was build according to [5]. One of the differences between full CNN and the shallow classifier is the definition of convolutional filters. In canonical CNN model, the coefficients of the filters are optimized while in the simplified approach they are fixed. In [5], as the filters act: vertical and horizontal bars, central parts of classified figures and the set trained for another model. At general, training networks for image recognition is very resource consuming task, thus reuse of existent results is generally a good idea.

In this task, the classifier was build using Haar wavelets and Legendre polynomials as for convolutional layer. Since the strides for convolution are larger than one, the side effect of the convolution is decimation, thus no explicit downsampling is needed. The results, however, were disappointing. Although for the train set the 2 layer hidden network of 128 or 256 neurons feed by the data filtered by above mentioned filters is was possible to get 90% match for train set, but applying the classifier to the test batch resulted in approximately 10% accuracy, which is what one could expect randomly labeling the images.

The results were calculated with scripts **shallow4.py** and **shallow_leg.py**. No improvement has been observed when some simple shapes (bar,dot) were used (**shallow_arb.py**). The conclusion is, that the filters in the shallow classifier, although set arbitrarily, must reflect some knowledge about the classified object.

4 Transfer learning

The aim of transfer learning is to utilize existing trained network for classification of another set of data. For this task, the *inception-v3* model has been chosen. The flow of the network is presented in Fig. 2. According to the figure, the data was passed directly to *ExpandDims* layer, in order to avoid jpeg decompression and integer to float conversion, since the CIFAR data are floating point. The output was taken from *pool_3* layer, which has the smallest possible size before the final classification. The model is trained to distinguish 2048 classes,

while only 10 are needed.

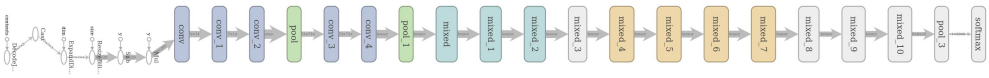

Figure 2: Inception-v3 model graph.

The first part of the toolchain is to process the CIFAR data through the the inception-v3 graph (**inception_features.py** script). It takes the image data and outputs a large csv file (approx 300M) per batch. As expected, the response of neurons responsible for activating several of 2048 was generally much stronger than others.

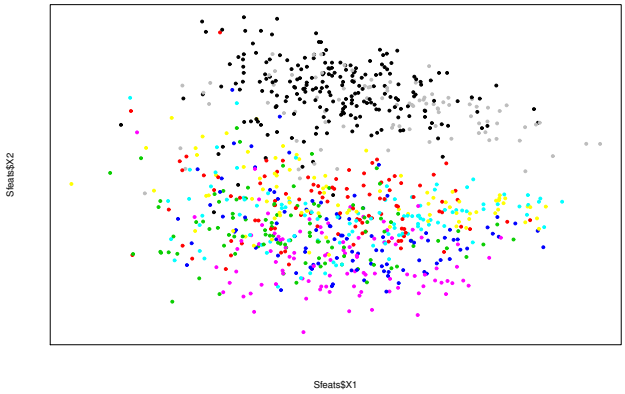

Figure 3: Placement of selected train images (groupped by color) depended on first two singular vectors.

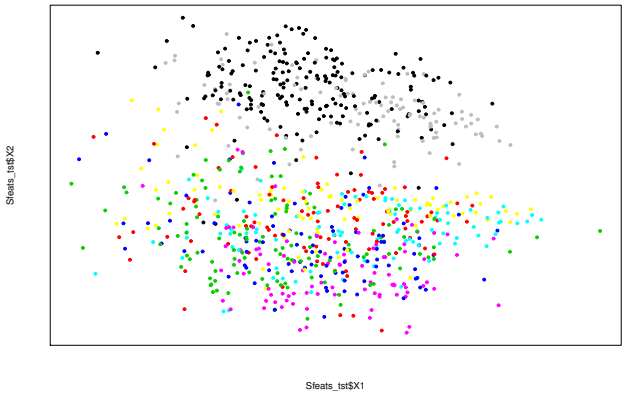

Figure 4: Placement of selected test images (groupped by color) depended on first two singular vectors.

In order to, at least partially, visualize so large set, two vectors, corresponding to two biggest singular values were chosen as coordinates. As seen in Figures 3 and 4, the colors (corresponding to different labels) are partially separable. This observation resulted in an idea to use the singular vectors instead of direct *inception-v3* output. This approach, however,

resulted in a very poor result: the hit rate for the test batch did not exceed 60%. On the contrary, when all 2048 features were used for svm classification, the accuracy was as large as 87% (1277 misses for 10000 cases). The result were calculated using **inception_svm.py** script. Linear classifier was used due to stability problems with other kernels.

5 Concluding remarks

For this task, very helpful tutorials were [1], [2]. The most creative part was training the shallow classifier, unfortunately with no great success. Changes suggested in section 3, however, namely better choice of filters should result in significant improvement. It would require spending more time, and some human interaction in order to get some important and useful heuristics. All scripts are put in HTTPS://GITHUB.COM/CEG-B/CIFAR-TRAINING

References

[1] Tenorflow Documentation *https://www.tensorflow.org/api_docs/python/tf*

[2] Tensorflow tutorial *https://www.kernix.com/blog/image-classification-with-a-pre-trained-deep-neural-network_p11*

[3] Alex Krizhevsky: *Learning Multiple Layers of Features from Tiny Images*. Tech report 2009.

[4] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, *Rethinking the Inception Architecture for Computer Vision*, arXiv:1512.00567, Dec 2015.

[5] Mark D. McDonnell, Tony Vladusich: *Enhanced Image Classification With a Fast-Learning Shallow Convolutional Neural Network*, arXiv:1503.04596, Aug 2015.